Technical Document 2635
30 September 1993

C³ Domain Analysis

Lessons Learned

R. M. Holmes
S. D. Rotter
S. A. Parker

308

NRaD

Technical Document 2635
30 September 1993

C^3 Domain Analysis

Lessons Learned

R. M. Holmes
S. D. Rotter
S. A. Parker

NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152–5001

K. E. EVANS, CAPT, USN
Commanding Officer

R. T. SHEARER
Executive Director

ADMINISTRATIVE INFORMATION

This work was performed for the Office of Naval Research, Arlington, VA 22217–5000. The work was performed under accession number DN088690, project ECB3, program element 0602234N.

Released by
R. E. Johnston, Head
Systems Branch

Under authority of
R. B. Volker, Head
Advanced Concepts and
Systems Technology Division

LH

CONTENTS

FIGURES

1.0 INTRODUCTION

1.1 ABSTRACT

This report describes lessons learned from the first phase of a domain analysis research project performed for the Office of Naval Research. The goal of this project is to evaluate the current state of the art in the emerging field of domain analysis and employ modern domain analysis methods in an area within command, control, and communications (C^3). Based on initial research, domain analysis is not yet a stable field, and appears to be evolving toward new methods being developed for systems analysis. Object-oriented methods appear the most popular at this time.

Domain analysis has been identified as a key technology in enabling systematic reuse. Despite its original promise, domain analysis has encountered several problems that have prevented its wide adoption by existing systems developers. The principal technical problems associated with performing domain analyses are the lack of standards and tool support. The primary nontechnical barriers to performing a domain analysis are "stovepiped" systems development and security restrictions.

Based on our research, we conclude that, at the present time, domain analysis can be used to support systematic reuse within Navy C^3 in narrow domains that have sponsors committed to reuse. There is little support evident for systematic reuse crossing sponsor boundaries in the Navy C^3 domain except for occasional opportunistic reuse. The lack of widespread support for systematic reuse will make the successful application of domain analysis throughout C^3 unlikely in the near future.

This task will continue research in domain analysis and begin investigating methods used in the implementation phase of domain engineering. The goal of this next phase is to implement an object-oriented domain analysis method linked to a "software architecture" capable of supporting C^3.

1.2 OVERVIEW

This report is the third in a series of reports describing the application of several popular domain analysis methods to the C^3 domain. Since C^3 is a broad domain encompassing many different fields, only a subset of C^3 was analyzed. This subset was the C^3 "subdomain" of message processing. Domain analysis is the systematic representation of information about a particular application domain with the aim of reusing that information across multiple applications within that domain. One purpose of this report is to provide feedback and lessons learned on our application of modern domain analysis methods. Perhaps a more important goal of this report is to suggest directions for future work to improve upon current domain analysis methods.

1.2.1 Domain Analysis Background

Domain analysis is currently viewed as a key ingredient in DoD's software technology strategy to encourage the systematic reuse of the tremendous amount of software and related "assets" that DoD produces each year. Although domain analysis is recognized as essential to the success of systematic reuse, it is not the only key factor. As the methodology of domain analysis matures, it is increasingly evident that other nontechnical factors are also essential to the success

of systematic reuse within DoD. These factors will be discussed in this report when the factors have had substantial impact on our application of domain analysis.

The emerging view is that there are few incentives and effective mechanisms to achieve any broad level of software reuse within the entire DoD. The main DoD reuse development effort is sponsored by the Corporate Information Management (CIM) program within the Defense Information Systems Agency (DISA). The CIM program is currently attempting to develop some uniformity among the myriad of independently developed information systems across the services. The CIM effort funds the Defense Software Repository System (DSRS) reuse library. At this time, however, there is no requirement across DoD to adopt the CIM standards and use their reuse library. Thus, the success of the CIM effort is not yet determined.

The Advanced Research Projects Agency (ARPA), primarily through the Software Technology for Adaptable Reliable Systems (STARS) and Domain Specific Software Architectures (DSSA) programs, provides the main research thrust for domain analysis and reuse technology.

The Air Force has two programs in reuse. The Central Archive for Reusable Defense Software (CARDS) and the Portable Reusable Integrated Software Modules (PRISM) programs are managed by the Electronic Systems Command. The Army is sponsoring reuse through its Common Ada Software System (CASS) and Reusable Ada Packages for Information Systems Development (RAPID) programs. The success or failure of these efforts will likely determine the future of systematic software reuse within the DoD. Currently, the future of these systematic reuse and domain analysis efforts is increasingly cloudy given the level of funding for these efforts and the constriction of the DoD budget.

Still, domain analysis is beginning a maturation process and does offer the prospect of providing badly needed structure to the current chaotic software development processes evident throughout DoD. This report will describe, in modest detail, the lessons learned from our application of several domain analysis and engineering methods.

1.2.2 Domain Analysis Methods

The STARS Reuse Library Process Model (RLPM) and the Software Engineering Institute's (SEI) Feature-Oriented Domain Analysis (FODA) [Cohen, et al., 1990] method were the original two methods used to analyze the C^3 subdomain of message processing. At the time our research began, these two approaches were the best known. Two other domain analysis tasks also affected the results of this report. They are the DoD's Corporate Information Management (CIM) domain analysis task [SofTech, 1993], and the ARPA DSSA command and control (C^2) task [Braun, et al., 1993]. Both of these tasks produced object-oriented domain analysis methods based on existing object-oriented methods traditionally used for systems analysis and design. Since these two domain analysis tasks are of a more recent vintage than RLPM and FODA, it is worthwhile to compare the older and newer methods to establish the general direction of evolution in domain analysis methods. Our examination and use of portions of these four methods provide the basis for this report.

Readers are directed to earlier reports [Rotter, et al., 1992, 1993], which describe the application of these methods in more detail. The main aim of this report will be to identify deficiencies in the methods used and then, later in the report, to propose solutions to those deficiencies. There will be relatively little exposition in this report about the specific models developed; rather the focus is on problems associated with the development of domain analysis

products, models, and taxonomies, along with recommendations on how these products and processes may be improved.

This report also distinguishes between descriptive models, which describe information about the "problem" space or domain of interest, and prescriptive models, which focus on "solution" space or computer science related aspects that are required to implement a system in the domain. In both cases, we describe how our analysis related to the aforementioned domain analysis methods, the problems we encountered, and recommendations for improving the methods.

The organization of this report is described in the next section. This organization is based on the fact that domain analysis experts usually make the separation of "what" is performed from "how" it is performed. Some experts have restricted domain analysis to only the "what" or problem definition phase of the analysis. The "how" or implementation issues are usually deferred to the later "domain engineering" phase. Domain engineering is also sometimes used to describe the entire process from problem definition to solution implementation. To avoid confusion regarding the definition of domain analysis and engineering, we will adopt the term "descriptive phase" to describe products and processes associated with the problem definition and analysis phase of domain analysis; the term "prescriptive phase" applies to processes and products associated with the implementation phase, in which reusable components are developed.

1.3 DOCUMENT ORGANIZATION

This document contains five sections, including this introductory section. Section 2.0 presents lessons learned during the descriptive phase of our domain analysis. Typically, the first models developed during a domain analysis are descriptive. Descriptive models provide representations of information about the domain of interest, in our case message processing, and concentrate on identifying what functions the systems in the domain are required to perform. These descriptive models are used mainly to provide structure to the domain and organize the domain to identify commonality among systems within the domain. Three classes of descriptive models are discussed separately: functional, dynamic, and object. The section is further divided into subsections concerned with domain analysis products and subsections associated with processes and procedures. There are also subsections providing recommendations for improving the analysis products and processes.

Section 3.0 is organized in the same manner as section 2.0, but covers prescriptive or implementation products and processes. The prescriptive models are split into two classes: software architectures and object-oriented designs.

Section 4.0 deals with the tool requirements of domain analysis and engineering. Current tool support is discussed along with tools to support the descriptive and prescriptive phases. In addition, tool requirements to develop systems are also briefly discussed. Normally, application generation or system composition are not considered part of domain analysis and domain engineering. However, one substantial barrier to reuse adoption in general and domain analysis in particular is the lack of tools to enable the development of systems using the products of domain analysis and engineering. Therefore, we have also included requirements for this class of tools.

Section 5.0 provides conclusions on the status of current domain analysis approaches and discusses recommendations for improving domain analysis methods.

2.0 DESCRIPTIVE PHASE—DOMAIN ANALYSIS

The first phase in most domain analysis methods is a descriptive phase. This phase determines the boundary of the domain and characterizes the important elements in the domain. RLPM accomplishes this goal with the development of a classification scheme and a taxonomy used to organize the domain. This approach is heavily based on the principles of library science and is geared toward a reuse effort with a large library-like repository of software and software-related components. In this approach, domain analysis is used to support the organization of components and their retrieval. Because RLPM begins with an examination of existing systems, it also contains some elements of the prescriptive phase. The classification and taxonomy factors in RLPM need not be only problem specific. Many components will be classified based on their implementation characteristics.

The CIM analysis method (based on Coad and Yourdon's object-oriented analysis) [Coad & Yourdon, 1991], the DSSA C^2 analysis method (based on Rumbaugh's object-oriented analysis method), and the FODA method do make the distinction between descriptive and prescriptive phases. Based on our examination of the methods, we believe that FODA and the DSSA domain analysis methods focus more on products than process, while RLPM and the CIM method focus principally upon the domain analysis process. All methods, however, describe both processes and products and only the relative emphasis differs.

2.1 DOMAIN ANALYSIS PRODUCTS

We have split the products of the descriptive phase into three groups: functional models, dynamic models, and object models. We selected this division because the distinction between functions, control, and data is made in all of these methods. Methods that emphasize functions or processing use functional models most heavily. On the other hand, object-oriented methods focus first on the data, data structures, and relationships between data structures. Dividing the products into the three groups allowed a better comparison between the methods.

Of the methods explored, only the RLPM with its bottom-up approach did not propose specific descriptive models. The primary form of organization proposed by RLPM was based on the classification scheme contained in the domain taxonomy. This was a drawback to the RLPM. In our estimation, a taxonomy provides insufficient structure to provide a framework for reuse. While it certainly supports a reuse library, the taxonomy does not provide enough information to determine how components can and should be connected to form working programs. Linkages and functional connections within the domain and between the domain of interest are also required and need to be captured in the domain analysis. While RLPM mentions constructing such models, it does not specify a methodology for descriptive models.

The remaining methods used a variety of models. The actual models used in the C^3 domain analysis were most closely tied to FODA. We will discuss the models within the three groups separately since different methods were more effective in different areas.

2.1.1 Functional Models

Functional models are used to describe the behavior of systems within a domain. This description is primarily from a user perspective. Our use of functional models coincided roughly with FODA. FODA uses two sets of descriptive models, context and domain models, and one set of prescriptive models, architecture models.

FODA context models consist of a context diagram and a structure diagram. The context diagram is essential in bounding the domain and is essentially the same diagram as the Yourden, Constantine Data Flow Diagram's context model. It describes external interfaces to the domain of interest. This diagram was quite useful in the C^3 domain because it allowed the separation of C^2 from the communications domain. The message processing subdomain of C^3 was contained within the C^2 subdomain. These context models allow for a clear representation and allow the assignment of functions to domains. In general, the context model was satisfactory and worked well in the message processing domain.

The second FODA context model was the structure diagram. The purpose of this diagram is to display other domains of interest. This chart proved to be somewhat difficult to interpret. However, the idea of interrelating domains is crucial to domain analysis. A typical C^3 application will deal with databases, user interfaces, input/output devices, and a variety of languages and operating systems. Each of these areas is itself a domain. The interfaces and interactions between these domains is one of the principle software development problems faced today. We selected an ad hoc method based on the Schlaer-Mellor object-oriented analysis method to represent the supporting domains for C^3. The area of describing domain interactions is one in which none of the methods available today is satisfactory.

FODA provides the feature model to show user-visible characteristics and describe system variability from a user perspective. We did not use this model, primarily because the feature model did not appear to scale for the message processing subdomain. Given the variety of C^3 message processing applications, constructing a useful feature model would be a large undertaking. Instead, we constructed a RLPM taxonomy to characterize the possible variability. The taxonomy approach appeared better able to scale up to be useful in moderate to large domains.

The features model appeared most useful in stable domains. Unfortunately, C^3 is stable in some subdomains, such as some parts of message processing, but volatile in others, such as user interface. In a volatile domain with many different components available and little standardization, the feature model is likely to rapidly become enormous as the domain evolves.

We performed additional functional modeling using a data flow approach We used the decomposition feature of the data flow method to produce a hierarchical decomposition of C^3 and message processing. This approach lent itself to automation using a CASE tool such as Software through Pictures. Unfortunately, this approach also lacked a rigorous foundation, particularly in precisely defining the interfaces between processes in the data flow diagrams.

For functional modeling, the DSSA C^2 domain model and the CIM domain model use the process modeling technique Integrated Computer Aided Manufacturing (ICAM) Definition (IDEF), developed by SofTech. DSSA used IDEF for functional modeling while CIM used IDEF to model the domain analysis process. IDEF models have been primarily used to model business practices and somewhat less often to model software processes in a manner similar to data flow modeling. IDEF has the advantage of being amenable to automation and can be rigorous. The principle problem we identified with IDEF is in comparing the results of the DSSA IDEF models to other models. The IDEF Context (top-level) model does not identify external domains, and IDEF does not have any concept of domain partitioning. It aims to specify what process needs to be performed. The allocation of processes and data (objects) to domains is not performed. In general, if an IDEF process is mentioned, it is in the domain of interest. Information and data coming into the domain is not classified or categorized by the domain that

sent the information. Similarly, information leaving the domain is not sent to another domain; rather the data are just sent out to another process. This lack of a domain orientation makes it difficult to recommend IDEF for functional modeling in domain analysis, however popular it may be for business process modeling.

Our discussion of the CIM use of IDEF to model the domain analysis process is deferred to the section on process modeling, although we can state here that the use of IDEF to model the domain analysis process appears more effective than its use in functional modeling.

2.1.2 Dynamic Models

Dynamic models describe the behavior, control, or temporal sequence of activities of a system. The use of these dynamic behavior-describing models varies considerably between the methods. Of the methods studied, the DSSA domain model for C^2 had the most detailed dynamic model. The DSSA dynamic model was developed using the Requirements Driven Design -100 (RDD-100) tool to present the dynamic behavior of a typical C^2 system. Discussing this tool is beyond the scope of this report, but the tool seemed to provide sufficient capability to model the flow of control of a system. Unfortunately, there are no standard representations for dynamic models. FODA used the product State-Mate to produce finite state-machine diagrams representation of objects within the domain of interest. The finite state-machine representation is used in other analysis methods also, notably the CIM domain analysis and the Schlaer-Mellor Object-Oriented Analysis technique. Generally, the finite state-machine representation seems preferred by the object-oriented methods.

In the message processing domain, the behavior, control, and sequencing of activities runs the gamut from totally prescribed to totally arbitrary. In an embedded message processor, the sequence of function calls is typically fixed for the system, while in a user-oriented text message editing system, the user decides the sequence of activities called. Because the processing sequence is quite variable across message processing systems, our approach was to create a separate "executive" process that handled the control and sequencing of the other functions. The FODA method seemed to be the method closest to the one employed in our analysis since it also recommended setting up executives to handle system control. This involved isolating the control from the functions in the message processing domain. This was done since the dynamic or control behavior of the message processing domain was the most volatile portion of the message processing domain. Since most portions of message processing are fairly stable, we performed the least modeling in the control area, preferring to expend our effort in characterizing the other more stable parts of the domain.

2.1.3 Object Models

Object models provide the biggest discrepancy between the older and newer models. There is little doubt that the older domain analysis methods have a heavy emphasis on functional models and the dynamic behavior of the systems. The object models produced using the older methods were largely geared toward database development and used entity-relationship models or some variation of the entity-relationship model.

Originally, FODA used the entity-relationship model to describe objects. Recently, however, Sholom Cohen, one of the principal developers of the FODA method, began experimenting with more object-oriented representations. Notably, he used a tool called 001, developed by Hamilton

Technologies, to represent the objects in his examples. In our domain analysis, we have also used 001 to perform object modeling. It provides a good way to model objects that are either part of another object (a "has a" relationship) or objects that are specialized forms (an "is a" relationship) of other objects. The 001 tool is a powerful data modeling, structured programming, code generation tool primarily designed to support application generation. Thus, 001 has a place as a tool supporting reuse, but is somewhat misused when applied to domain analysis and object modeling.

The newer domain analysis methods use object models lifted from the latest object-oriented analysis techniques. CIM proposes using Coad-Yourden Object-Oriented Analysis diagrams, while DSSA uses the Rumbaugh Object-Oriented Analysis method and diagrams. These methods have the benefit that popular CASE tools, such as CADRE's Teamwork, IDE's Software through Pictures (StP), and Rational's ROSE support the production of such object-oriented diagrams. In Teamwork and StP, these object diagrams can be linked to the actual code implementing the object. This supports a tight integration of the domain model produced by the domain analysis and the software implementation. This integration offers a solution to one of the primary technical problems in domain analysis, namely, how to link the problem space analysis models to the solution space implementation models. As our own experience with these object-oriented methods is quite limited, we will not discuss the similarities and differences between the various object-oriented methods available today. Evaluation of these methods will, however, be performed later in this task.

So far, we have not implemented object models using an object-oriented analysis method. This will be a natural next step, given the obvious trend toward object-oriented analysis, design, and programming. We have also not discussed the drawbacks in using object-oriented analysis methods to perform a domain analysis. The chief drawback in using current object-oriented methods for domain analysis is that these methods were designed to support systems analysis. The goals of systems analysis are to support the design and development of a system. These methods are not designed to model variability; in fact, they are intended to produce an unambiguous and, hence, nonvariable result. These methods typically drive the analysis toward a single or at least limited solution and implementation, rather than to characterize all the possible solutions to the problem.

In some sense, the shift to object-oriented methods indicates that the field of domain analysis is backing off from attempting to describe all possible variations in the problem and solution domains, and, instead, restricting both the problem and solution space to a small manageable number of alternatives. This restriction is definitely required in a field such as C^3. The variation in function, behavior, and objects in existing systems alone is staggering, and attempting to classify all the systems in use today would be a massive undertaking. Clearly, what is needed is to restrict the scope of functions, behavior, and objects right from the start of the domain analysis. This restricts the variability that can be described, but offers the hope that the domain can be organized under a single or small number of domain models. The goal of these models is to adequately describe most systems and to be used as the standard for the enhancement of existing systems and the development of new systems. These models would provide a basis to reengineer all existing systems not following the standardized domain models.

2.2 IMPROVING DOMAIN ANALYSIS PRODUCTS

There are five primary areas requiring improvement in domain analysis. They are:

Standards
Representations
Tools
Object-oriented methods
Definitions and goals of domain analysis

The lack of standards for domain analysis products remains a serious impediment to the adoption and use of any of the current domain analysis methods. Naturally, the lack of standards is due to the wide variety of competing methods all using different, and incompatible, representations. Until adequate representations are developed, it is likely that little standardization will be possible.

The key technical area facing domain analysis is to provide an adequate representation of domain information. An adequate representation must provide a precise, unambiguous description of the problem space, capable, ultimately, of generating or composing systems or subsystems. At this point, adequate representations only exist in narrow service domains, such as the relational database domain and the user interface domain.

In general application domains, however, the choice of tools usually determines the representation. That is, if a CASE tool is available, the domain analysis is performed using the CASE diagrams. The development and maturation of the domain analysis methods has been significantly slowed by the lack of tools specifically designed for domain analysis. Tools designed to support domain analysis are needed. These tool requirements are discussed more fully in section 4.0.

Object-oriented methods offer the possibility of supporting the key requirements of domain analysis. Current object-oriented methods provide a precise representation of the problem space and "map" in a straightforward way to the solution space. There are currently many object-oriented methods in use today. Since they are much more widely used than domain analysis methods, the number of popular methods in wide usage should begin to reduce to a few methods, supported by the most popular CASE tools. An effective domain analysis method should arise from at least one of these object-oriented methods. This object-oriented method will be supported by CASE tools and, hence, provide an adequate representation of systems in the domain.

For object-oriented methods to provide a satisfactory solution for domain analysis, domain analysis will have to revise its own goals and definition. Currently, domain analysis attempts to categorize the variability of possible and existing systems in the domain in addition to providing the framework and structure to build multiple systems within the domain. If the definition of domain analysis is revised to only providing the framework and structure to build a restricted family of systems within the domain, object-oriented methods provide a nice fit. This change in definition restricts the scope and purpose of the analysis but provides a better means to implement an architectural solution. This is not to say that categorization of the variability of systems cannot be done; rather, once the variety of systems is described, restrictions on that variety are soon made to limit the scope of possible systems that can be implemented. The domain model

then represents a restricted subset of systems with the goal to provide a framework for building systems within that restricted subset. Although not explicitly stated, support for this notion of restricted reuse is present even in the earlier methods [Prieto-Diaz, 1991]. In Prieto-Diaz's method, the systems are categorized and cataloged, then a "shrinkage" occurs. In this shrinkage, systems and components are evaluated for commonality and extraneous or aberrant components or systems are discarded. The result is a reduced set of components, which are then reengineered to a common but more restrictive framework.

To summarize, our suggestions for improving domain analysis processes are consistent with recent trends in domain analysis research. First, adopt currently popular object-oriented analysis methods. Switch from a predominately functional view of systems to an object-oriented view. Use CASE tools implementing object-oriented methods to represent the domain analysis results. The domain model resulting from this object-oriented approach will no longer focus on the variability of the domain. Instead, the domain model will provide a restricted framework for the domain, based on the objects in the domain. As object-oriented methods stabilize, standards will evolve. Under this scenario, domain analysis will merge with object-oriented analysis. In some ways, this merger is already beginning. The emphasis in object-oriented methods has been steadily moving toward the creation of reusable class libraries. Object-oriented analysis then provides the framework for connecting these reusable classes. This framework satisfies the main requirement of the restricted view of domain analysis.

In the research performed so far on this task, we would recommend using the Schlaer-Mellor object-oriented analysis method [Schlaer & Mellor, 1989] to perform a C^3 domain analysis. The primary reasons for this recommendation are twofold. First, the Schlaer-Mellor method provides the strongest domain orientation of all the current object-oriented methods. In fact they devote an entire chapter of their book "Object Lifecycles, Modeling the World in States" [Schlaer & Mellor, 1992] to a discussion of domains. Their notion of a domain is consistent with domain analysis and they propose the novel concept of "software bridges" to connect domains.

Second, the Schlaer-Mellor method is also among the most detailed and complete, including both process and products. The method also includes tight connections between the analysis phase and reusable component implementation using their recursive design technique. Since this method has not yet been applied as a domain analysis method, one recommendation of this report is to perform a Schlaer-Mellor analysis and recursive design implementation on the message processing domain (or other suitable C^3 subdomain).

2.3 DOMAIN ANALYSIS PROCESSES

The size of the effort described in this report precluded the use or adoption of any extensive domain analysis process. Two methods, the RLPM and the CIM methods, provide detailed process models that appear reasonable for larger scale domain analyses. The applicability of these processes to C^3 is discussed in a later paragraph. The FODA method was primarily product-oriented with little process advice.

The process model followed in this task was originally suggested by Prieto-Diaz and, subsequently, partially included in the RLPM. His basic recommendation was to first sift through existing systems, including source code and documentation, to identify commonality. In the message processing area, we were moderately successful, while in the C^3 area, this proved essentially impossible. The fundamental reason for the failure of the domain analysis process

across C^3 was the "stovepipe" nature of current systems development. Within Navy programs, information about the program is usually available to those working on the program. Across programs, however, the situation changes dramatically. In some cases, the program sponsor restricts access to source code of current systems, while in others, little documentation is available. In many other C^3 systems, much of the information has security restrictions, making it difficult to use in this task. The result was that the overall C^3 information used to construct the domain taxonomy and models for C^3 were constructed from more high-level architectural documents rather than actual fielded C^3 systems. Thus, the resulting C^3 model remained at a relatively high level and represented more conceptual models for C^3 that are quite coarse-grained in their coverage of C^3. These models lack the technical detail that would come from examining real fielded systems.

The message processing models, however, used more detailed documentation and existing unclassified systems. This produced a more detailed taxonomy for the message processing domain. Even within the message processing domain, knowledge of available systems was frequently limited to systems organizationally "close" to the task members. This makes it difficult to say that our domain analysis is representative of systems throughout the Navy, much less the other services. Still, at least one of the key systems used, the Joint Automated Message Editing System (JAMES), has been used for several years in preparing message text format (MTF) messages in the joint arena.

Unfortunately, most of the process models proposed today assume an open organizational structure with access to diverse systems easily available within the organization. For the foreseeable future, the Navy, and, presumably, the other services do not operate this way. Software development in the C^3 domain is divided into many competing subdomains that have little interest in sharing knowledge or resources. Unless there is a dramatic shift in the organizational structure that develops C^3 systems, the application of any large-scale domain analysis process seems unlikely to have substantial impact on fielded system development. This, of course, also dims the outlook for substantial systematic reuse in the C^3 domain. There is encouragement, however, in that at the sponsor level, cost is driving many program executive officers and program managers to begin consolidating systems and eliminate redundant functionality. The Navy's shore and afloat tactical C^3 sponsorship, for example, already identified common "core" components that can be used as building blocks for current and future tactical systems. This effort can be viewed as a domain analysis within that one sponsor's programs.

The final area of concern about domain analysis processes is the level of support they offer to evolutionary system development. Evolutionary system development is emerging as the most popular way to build systems in an era of rapid technical development. The idea behind evolutionary development is that technical progress cannot currently be accurately predicted over the 20-year expected lifetime of current systems. In order for systems to adapt to new technical developments, the system development process must frequently reevaluate technical requirements and enhance or reengineer the system to operate under new conditions.

The need for an evolutionary development process has not really been recognized in current domain analysis processes. Although, most methods indicate that producing a domain model is not a one-time effort, few indicate how the domain model is to evolve over time. Thus, although there is a feedback loop in most processes, relatively little attention is given to the mechanisms and processes that support the feedback to the analysis models as the domain evolves. Without

allowance for dynamic evolution of a domain, these processes are relegated to stable unchanging domains. Unfortunately, C^3 is not such a domain.

2.4 IMPROVING DOMAIN ANALYSIS PROCESSES

The previous section and its discussion of the problem areas provides obvious areas and means for improvement. The biggest hurdle to overcome in implementing large-scale systematic reuse across a broad domain such as C^3 is organizational and, hence, process oriented. The current organizational structure and methods for developing C^3 systems precludes the wide-spread use of domain analysis and reuse processes and products available today. These methods can be effectively used within subdomains, particularly domains controlled by one sponsor. Extending domain analsyis across different sponsors will require considerable organizational and political pressure since any substantial reuse effort will divert resources from current development efforts.

Improving the domain analysis process to handle the evolutionary development should not be too difficult. In fact, if object-oriented methods continue to develop and the analysis and implementation phases are tightly connected, the evolution of the domain model will drive system evolution since the domain model will be used to generate or compose the next evolution in the systems.

3.0 PRESCRIPTIVE PHASE—DOMAIN ENGINEERING

The prescriptive phase takes the products of the domain analysis (usually referred to as the domain model) and translates or maps the model to a framework from which working implementations can be implemented in a straightforward manner. The framework is usually called a software architecture. This architecture should contain sufficient detail and processes for system design to proceed directly from the architecture. The development of a software architecture is beyond the traditional scope of domain analysis. Architecture development is typically included with domain analysis in the broader term "domain engineering."

This section briefly describes lessons learned so far in performing initial steps in the prescriptive phase of domain engineering, that is, developing software architecture and producing a framework for system design. The task reported here has only begun to enter this phase; hence, few products and processes have been tested. There are, however, some early results to comment on.

3.1 DOMAIN ENGINEERING PRODUCTS

The key product in the domain engineering phase is a software architecture, sometimes called a generic architecture, for the domain. Compared to the domain analysis phase, relatively little research has been published on the ingredients necessary in a software architecture to support domain engineering. From our brief experience in developing a software architecture for message processing and attempting to use that architecture to develop a system design, we already can provide some feedback in this area. This will be a key area in our future efforts.

3.1.1 Software Architectures

Conventional software architecture representations seem to have standardized on a diagram known as a reference architecture. A popular reference architecture in widespread use today is the National Institute of Standards and Technology's (NIST) Application Portability Profile (APP). The NIST APP is displayed in figure 1. Within DoD, versions of this reference model have appeared in the CIM technical reference model.

This architecture describes a layered set of services, with an application level on top. The application layer contains a support applications layer immediately below. Both of these layers would vary depending upon the domain of interest. The remaining service layers would presumably support multiple domains. The services are generic in the reference model. Specific standards for the services are proposed in a reference model standards profile. The specific standards typically include both current standards and proposed future standards.

In our task, we developed a reference model supporting the C^2 domain. It is displayed in figure 2. This architecture fills in recommended support applications for message processing and adds another domain-tailored core service layer below the domain-tailored services (support applications) layer. In the bottom layers, this architecture is consistent with the NIST APP.

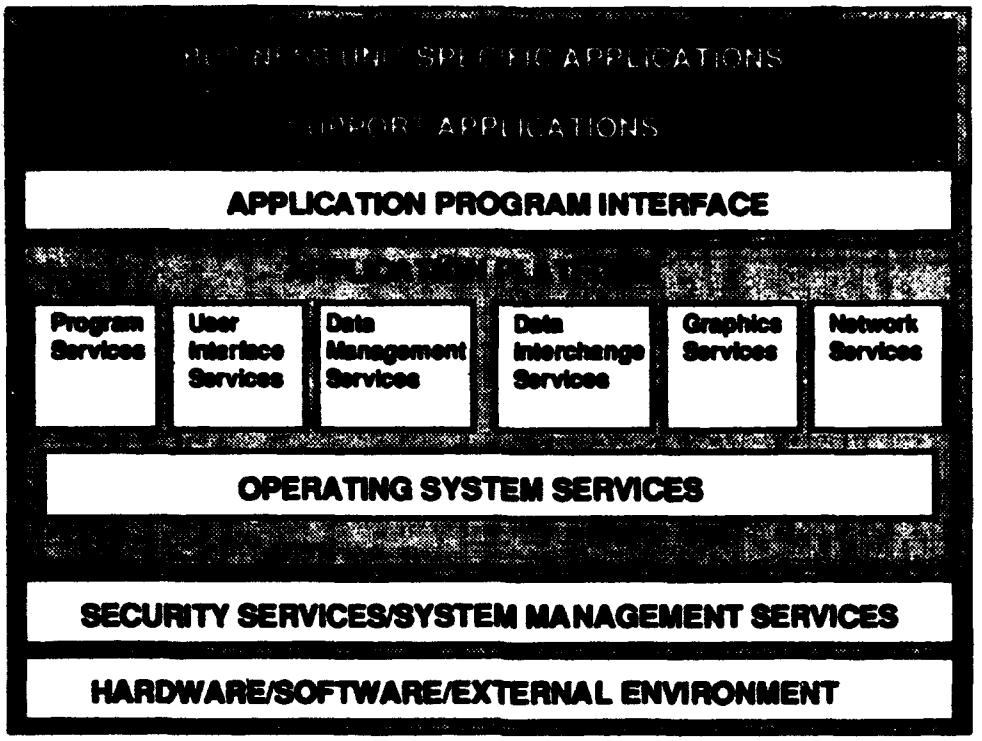

Figure 1. NIST application portability profile reference model.

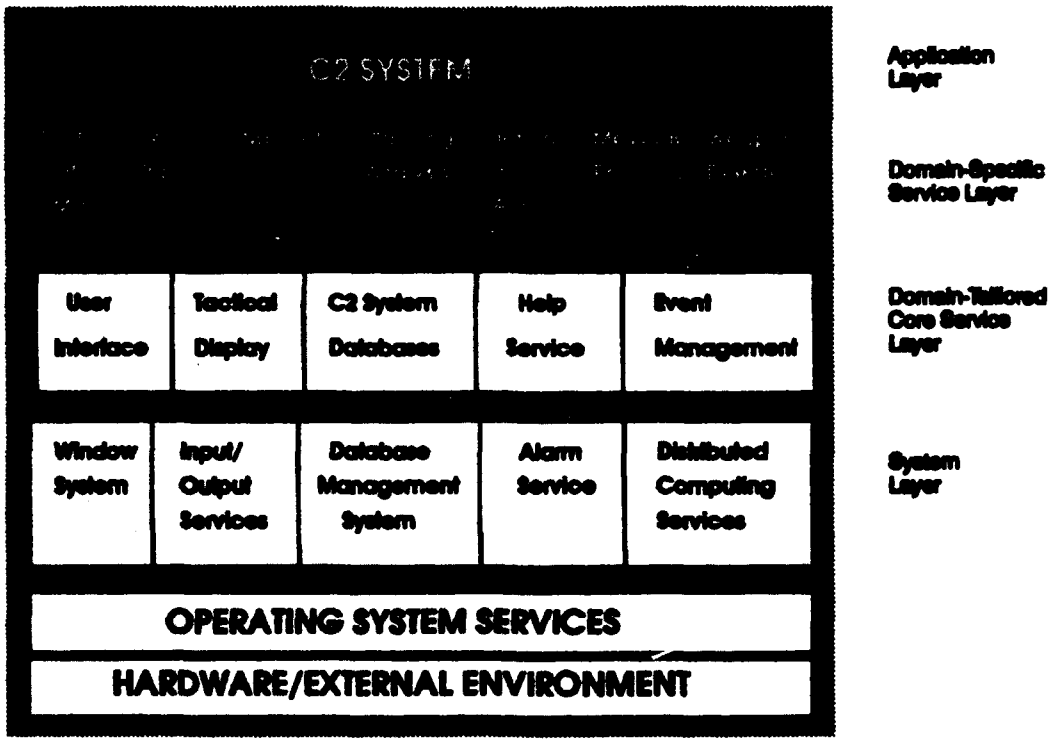

Figure 2. Command and control reference model.

In this initial development, several problems have already surfaced. The purpose of the software architecture is to serve as a bridge between the results of the domain analysis (the domain model) and the system design. In fact, it appears that a reference architecture does neither. First, there is no mapping back to the domain model and, second, the reference architecture provided insufficient insight into how systems are to be constructed using it. Certainly, if one used components developed using the standards prescribed by the reference model standards, the system will satisfy the reference architecture. This, unfortunately, does not prescribe the system design very much. A useful software architecture should lead to, at most, a few possible system designs. Clearly, more is required than simply a reference architecture to provide a framework to support reuse.

3.1.2 System Design

In general, most reference models do not contain enough information to produce an unambiguous design (object-oriented or not) for a given system. This is mainly due to the lack of detail described earlier in current reference models. Also, based on our recommendation to use object-oriented analysis methods, object-oriented designs (OODs) would be the obvious choice to use in system design. Unfortunately, current software architectures, being functionally or service oriented, do not particularly support OOD.

3.2 IMPROVED DOMAIN ENGINEERING PRODUCTS

Our main suggestion for improving software architecture is to add more detail. A reference architecture is insufficient to provide the necessary linkage between the domain model and system design. What is needed is a generic system design with prescribed and well-defined interfaces. The generic units must be mapped unambiguously back to the domain model, either functionally or by objects. When the domain model changes or evolves, the generic system design must have some defined way to change or add new design elements and their interfaces. Some of the more recent object-oriented analysis and design methods are close to providing this linkage. To date, the more functionally oriented domain analysis methods have had a difficult time maintaining the correspondence between the domain model and system design. Tools to automatically "trace" changes in the domain model through to the software architecture are not available, probably because there are no emerging standards for domain models, for reference architectures, or for how this traceability is to be achieved.

4.0 TOOL REQUIREMENTS

4.1 CURRENT TOOL SUPPORT

There is little current tool support geared specifically for domain analysis and domain engineering because the field is still too small and immature to have much in the way of specialized tools available. Fortunately, much of the work in domain analysis is being accomplished through the use of existing tools developed for other purposes. In this section, we will describe in tabular form, the general tool requirements for descriptive models, prescriptive models, and application generation. The tools are described only by general type and no specific products are proposed. Some of these tools were originally specified in the CECOM Report "Impact of Domain Analysis on Reuse Methods" [Software Productivity Solutions Inc., 1989]. Many of these tools are currently available, although they have not been combined into an integrated domain analysis toolset.

4.2 TOOL REQUIREMENTS FOR ALL PHASES OF DOMAIN ENGINEERING

Project management
Software cost modeling
Word processor and desktop publishing
Database management system
Configuration management and version control
Traceability
Change propagation

4.3 TOOL REQUIREMENTS FOR DESCRIPTIVE MODELS—DOMAIN ANALYSIS TOOLS

Functional or object modeling
Data dictionary
Specification language (graphical or text-based)
Clustering/cataloging (for Prieto-Diaz)
Knowledge capture

4.4 TOOL REQUIREMENTS FOR PRESCRIPTIVE MODELS—DOMAIN DESIGN TOOLS

Architecture design
Systems analysis
Reuse library
- Component cataloging
- Component classification
- Component qualification
- Component management

Simulation
Prototyping

4.5 TOOL REQUIREMENTS FOR APPLICATION GENERATION—COMPONENT DEVELOPMENT

Reuse library
- Component identification
- Component development
- Component tailoring
- Component storage
- Component retrieval
- Component integration

Simulation
Prototyping
System composition/generation
Testing
Software development environment/CASE
Reverse engineering
Reengineering

4.6 IMPROVED TOOL SUPPORT

The primary requirement for improved tool support is better tool integration. Standards such as the Portable Common Tool Environment (PCTE), which provides a framework and data interchange standards for the exchange of data between tools, and Object Linking and Embedding (OLE), a Microsoft standard for applications to use other applications, offer robust ways to share data and information among different tools. One area of additional work in this task will be to evaluate these different standards for use in tool and component integration.

Since domain analysis as a field is small, it will likely have to follow existing software development tool vendors rather than to attempt to define a domain analysis standard for tool integration and domain analysis knowledge representation. Being a front-end activity in the software development life cycle, domain analysis activities will most likely use front end CASE tools such as Teamwork and Software through Pictures to capture and represent most information. Special-purpose domain analysis tools will then be integrated into the CASE tool to provide capabilities uniquely required by domain analysis.

Many of the tools mentioned in the list, notably reverse engineering, reengineering, knowledge capture, and system composition or generation tools, are still themselves quite immature. Thus, they now provide only limited support for domain analysis. Further development of these immature tools will be required to adequately support domain analysis. As they improve and provide adequate representations for individual systems, these same tools can be used in domain analysis to develop the domain models.

5.0 CONCLUSIONS AND RECOMMENDATIONS

Domain analysis is a young field that has not yet been fully defined and scoped. The original view of domain analysis as a methodology to characterize families of systems within a domain and then to provide a framework to develop a wide variety of systems has evolved. Having been somewhat oversold, domain analysis seems unable to achieve this flexibility. The trend now seems to be toward restricting the framework and, thereby, reducing the variety of systems in the domain. This permits more reuse potential and greatly simplifies the development of an implementation framework. The precise representation of this framework using a software architecture is still an open problem that exists both in domain analysis and systems analysis.

Although many of the technical issues concerning representing domain information and tool support still exist, they can be solved provided domain analysis and systems analysis receive the support necessary to mature. Most likely, this maturation will involve the use of object-oriented analysis and design methods adapted from systems analysis. Since systems analysis is a "bigger" field than domain analysis, domain analysis standards will most likely be developed within the context of systems analysis.

Our recommendation is to concentrate on the popular object-oriented systems analysis approaches to perform domain analysis Among them, Schlaer-Mellor Object-Oriented Analysis represents a fairly complete and detailed approach. It has been proposed here at NRaD as a "standard" for use in systems analysis and design and should be effective in domain analysis also. The applicability of the Schlaer-Mellor method to perform domain analysis and engineering needs to be tested and validated. Finally, the Schlaer-Mellor method allows considerable latitude in the definition and representation of a software architecture. A satisfactory definition of software architecture that can be used to support the development of multiple systems within the C^3 domain needs to be developed and tested. This architecture needs to be compatible with modern reference models and software architectures such as the NIST APP, yet provide more detail in the application layer(s) about the software design and interface specifications between layers in the reference model.

6.0 REFERENCES

Braun C.L., et al. 1993. "Domain Specific Software Architectures, Command and Control, Domain Model Report," CDRL CLIN 0006, Contract DAAB07-92-C-Q502, GTE Federal Systems Division.

Coad, P., and E. Yourdon. 1990. *Object Oriented Analysis,* Yourdon Press Computing Press, Englewood Cliffs, NJ.

Cohen, S. et al. 1990. "Feature-Oriented Domain Analysis (FODA) Feasibility Study." CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA.

Prieto-Diaz, R. 1991. "Making Software Reuse Work: An Implementation Model," Software Engineering Notes, ACM SIGSOFT, Vol. 16, no 3.

Rotter, S.D. et al. 1992. "A Taxonomy for the Command and Control Domain," Draft Report Naval Command, Control and Ocean Surveillance Center Research, Development, Test and Evaluation Division, San Diego, CA.

Rotter, S.D. et al. 1993. "Domain Analysis and Engineering for the Command, Control and Communications (C^3) Message Processing Domain," Draft Report Naval Command, Control and Ocean Surveillance Center Research, Development, Test and Evaluation Division, San Diego, CA.

Schlaer, S., and S. J. Mellor. 1989. "An Object-Oriented Approach to Domain Analysis," *Software Engineering Notes,* ACM Press.

Schlaer, S., and S. J. Mellor. 1992. *Object Lifecycles, Modeling the World in States,* Prentice Hall, Inc. Englewood NJ.

SofTech, Inc., 1993. "Domain Analysis and Design Process, Version 1," Document Number 1222-04-210/30.1, DISA-CIM Software Reuse Program Office.

Software Productivity Solutions, Inc., 1989. "Impact of Domain Analysis on Reuse Methods," C04-087LD-0001-00, U. S. Army CECOM Center for Software Engineering, Ft. Monmouth, NJ.

Software Technology for Adaptable, Reliable Systems (STARS) Program. 1991. "Reuse Library Process Model." Electronic Systems Division, Air Force Systems Command, USAF, Hanscom AFB, MA.

APPENDIX A. GLOSSARY

Abstraction: An abstraction encapsulates essential object characteristics that distinguish it from all other types of objects and provides defined conceptual boundaries.

Application: A software program that provides a solution to some type of user problem.

Context: The circumstances or environment in which a particular system exists.

Context Model: A view of the problem domain that depicts the interaction between the domain under study and the connecting domains or other systems.

Descriptive Model: A view (or multiple views) of the problem domain that describes the entities in the problem domain, their interrelationships, and key functions performed in the problem domain.

Descriptive Phase: The portion of the domain engineering activity that produces descriptive models.

Domain: A family of current and future system applications that share common capabilities and data.

Domain Analysis: The domain engineering activity that identifies, organizes, and represents information from a family of systems (a domain) based on the study of existing system documentation and information that can be learned from domain experts.

Domain Analyst: The person conducting the study of existing system documentation and acquiring information through communicating with domain experts.

Domain Engineering: The activity encompassing domain analysis and domain implementation resulting in system development.

Domain Expert: A person with expertise in the domain under study.

Domain Implementation: The domain engineering activities that use the products of domain analysis to develop a new system.

Domain Model: A definition of the functions, objects, data, and related interfaces usually depicted graphically.

Dynamic Model: A description of the control of a system, particularly emphasizing the time-dependent processing and temporal ordering of functions.

Facet: An aspect of the domain used for classification.

Function: An operation on an object within a domain.

Functional Model: A depiction of functions, usually within the problem domain.

Object: A thing in the domain which is acted upon and acts upon other objects.

Object Models: A description of the objects in a system and the interelationships between objects.

Prescriptive Model: A view (or multiple views) of the implementation domain that describes the entities in the implementation domain, their interrelationships, and key functions required to produce an implementation.

Prescriptive Phase: The portion of the domain engineering activity that produces prescriptive models.

Reusable Component: A software component (not restricted to source code) specifically designed and implemented to be reused.

Software Architecture: A high-level depiction of functions, objects, control, data, and related interfaces to support the implementation of applications in a domain.

Software Reuse: The process of developing new software systems using existing software components.

Taxonomy: A collection or group of domain-relevant terms that provide a classification scheme for identifying and describing components within the domain.

APPENDIX B. ACRONYMS

APP	Application Portability Profile
ARPA	Advanced Research Projects Agency
C^2	Command and Control
C^3	Command, Control, and Communications
CARDS	Central Archive for Reusable Defense Software
CASS	Common Ada Software System
CIM	Center for Information Management
DADP	Domain Analysis and Design Process
DISA	Defense Information Systems Agency
DSRS	Defense Software Repository System
DSSA	Domain Specific Software Architecture
DoD	Department of Defense
FODA	Feature-Oriented Domain Analysis
ICAM	Integrated Computer Aided Manufacturing
IDEF	Integrated Computer Aided Manufacturing (ICAM) Definition
NIST	National Institute of Standards and Technology
OLE	Object Linking and Embedding
OOD	Object-Oriented Design
PCTE	Portable Common Tool Environment
PRISM	Portable Reusable Integrated Software Modules
RAPID	Reusable Ada Packages for Information Systems Development
RDD-100	Requirements Driven Design - 100
RLPM	Reuse Library Process Model
SEI	Software Engineering Institute
STARS	Software Technology for Adaptable Reliable Systems
StP	Software Through Pictures

APPENDIX C. DOMAIN ANALYSIS METHODS

This appendix contains a brief description of the domain analysis methods discussed in this report. For more detail, the reader should consult the references on each method.

FEATURE-ORIENTED DOMAIN ANALYSIS (FODA)

FODA was developed by Sholom Cohen and others as part of Domain Analysis Project at the Software Engineering Institute. FODA has three distinct phases; context analysis, domain modeling, and architecture modeling. The context analysis scopes the domain and establishes boundaries between the domain and other domains. The domain modeling phase constructs a description of the problem space called the domain model. Lastly, the architecture phase produces a software architecture sufficiently detailed to construct applications. FODA is primarily a model-based method and produces a variety of models corresponding to the three phases. A complete description of these models is beyond the scope of this appendix; however, these models are predominately functional models. The FODA method currently lists the following models.

Context diagram
Structure diagram
Domain terminology dictionary
Entity-relationship model
Feature model
Functional model
Object connection update model

FODA is currently being used in the Maneuver Control subsystem of the Army's Common Ada Software System (CASS).

STARS REUSE LIBRARY PROCESS MODEL (RLPM)

There are five steps in the STARS process model:

Knowledge acquisition
Domain definition
Classification and keywords
Functional models
Domain architecture

There are two key factors to note about the STARS domain analysis process. First, the STARS domain analysis is heavily biased toward process. There is little exposition on the specific models or domain representations that are to be produced. There is also little guidance provided on how to perform much of the process. Second, the STARS process is a combined top-down and bottom-up process. The top-down portion of the process is described mainly in the functional models with the bottom-up analysis performed in the classification and keywords analysis. This combined approach was unique to the RLPM. The RLPM is based on Ruben Prieto-Diaz's work. This method adapts many principles from library science to the organization and implementation of a reuse library.

DEFENSE INFORMATION SYSTEMS AGENCY (DISA) CENTER FOR INFORMATION MANAGEMENT (CIM) DOMAIN ANALYSIS AND DESIGN PROCESS (DADP)

The domain analysis process draft document proposes a fairly complete domain analysis process via IDEF diagrams. The activities described in the DADP roughly parallel the STARS RLPM. The description of the products, however, is much more detailed. Each of the models developed is illustrated by an example. The DADP also proposes using an object-oriented methodology. Most of their diagrams are based on the Coad-Yourden Object-Oriented Analysis diagrams and method, but there is no specific requirement in their guidelines to use any particular method. In fact, they reference most of the current object-oriented analysis methods, including Bailin, Booch, Rumbaugh, and Schlaer-Mellor. This method is among the newest of the group and is likely to undergo still more changes. The main advantage of this method is that it corrects much of the vagueness in the STARS process, particularly in model construction and definition.

ADVANCED PROJECTS RESEARCH AGENCY (ARPA) DOMAIN SPECIFIC SOFTWARE ARCHITECTURES (DSSA)—COMMAND AND CONTROL

The DSSA domain model consisted of three groups of models applied in a top-down fashion: functional, dynamic, and object. This method also provided the most novel application of IDEF, using it to represent the functional models rather than the domain analysis process. The domain model report contained little process information and concentrated mostly on describing the three models for the command and control domain.

The dynamic models were represented using Requirements Driven Design (RDD) - 100, developed by the Ascent Logic Corporation. This method is not widely known and describes functional processes in a manner similar to the standard data flow diagram. The method also allows control flows on the diagrams and indicates on these flows if the execution may be concurrent or sequential. Like the data flow and control flow diagrams, RDD-100 offers a hierarchical decomposition of the systems functions.

The object model uses the Object Modeling Technique (OMT) of Rumbaugh. From an object-oriented standpoint, the class representations are quite standard, using an object/class diagram containing a class name, class attributes, and class functions or methods. The relationships "is a" and "has a" are represented with different line connections between the classes. Relationships between classes are represented with the concept of an "association" between the classes. Associations may have a multiplicity, such as a vehicle has one or more wheels. The technique can display these types of associations.

The choice of object-oriented analysis diagramming method in the DSSA application seems flexible. It appears that any of the current object-oriented analysis methods contain diagrams capable of describing the information contained in the DSSA C^2 Object Model. Using a different method to display the C^2 object models would merely result in slightly different notation. Thus, the use of the Rumbaugh notation was not a requirement of their method; rather, it was more of a convenience.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	30 September 1993	Final

4. TITLE AND SUBTITLE

C^3 DOMAIN ANALYSIS
Lessons Learned

5. FUNDING NUMBERS

AN: DN088690
PROJ: ECB3
PE: 0602234N

6. AUTHOR(S)

R. M. Holmes, S. D. Rotter, S. A. Parker

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division
San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER

TD 2635

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Arlington, VA 22217–5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report describes lessons learned from the first phase of a domain analysis research project performed for the Office of Naval Research. The goal of this project is to evaluate the current state of the art in the emerging field of domain analysis and employ modern domain analysis methods in an area within command, control, and communications (C^3). Based on initial research, domain analysis is not yet a stable field, and appears to be evolving toward new methods being developed for systems analysis. Object-oriented (O-O) methods appear the most popular at this time.

14. SUBJECT TERMS

Domain engineering
Domain analysis methods
Command, control, and communications (C3)
Software reuse

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE (Include Area Code)	21c. OFFICE SYMBOL
R. M. Holmes	(619) 553-4079	Code 4128